Pipelining and Superscalar using SimpleScalar (1)

In this lab we shall implement pipelined and superscalar configurations on PISA ISA and compare the two. You can do the same for Alpha ISA. We shall also see if forwarding is being implemented or not.

We shall be using sim-outorder as the simulator to calculate IPC or CPI in various processor configurations.

Quick fact about sim-outorder simulator

It is a detailed microarchitectural simulator and models timing. This tool models in detail in-order and out-of-order microprocessor with all of the bells and whistles, including branch prediction, caches, and external memory. This simulator is highly parameterized and can emulate machines of varying numbers of execution units. Refer to reference #4 of introduction to SimpleScalar to get a visual description of this simulator. You should read the references to know the 5 stages in PISA architecture.

All the simulators including sim-outorder are available in the home/simplesim-3.0 directory.

Go to home/simplesim-3.0 directory and type the following to seek help about sim-order.

```
home/simplesim-3.0$./sim-order –h
```

Help can also be invoked just by typing simulator name without any arguments.

1) In this part, we shall see whether sim-outorder implements forwarding to reduce stalls due to data hazards or not.

Follow the procedure described below.

a) Create new directory e.g., labs in the /home/SimpleScalar directory. This is where you shall keep all the files
for this lab.

b) Copy default.cfg configuration file from /home/SimpleScalar/simplesim-3.0/config/ into the labs directory.

c) Make a copy of the file default.cfg and name it e.g., config_a.cfg.

d) Copy test-math binary file from /home/SimpleScalar/testsimplesim-3.0/tests-pisa/bin/ into the labs directory.

e) View config_a.cfg file in an editor or text viewer. This file is used by sim-outorder simulator. The file is easy to understand. To dig into the details, refer to reference #2 of introduction to SimpleScalar.

f) Modify config_a.cfg file according to the following specifications.

Decode:width = 1,
Fetch queue size = 1,
Load store queue size = 8,
Register update unit (ruu) size = 8, //This is the same as the number of reservation stations
Issue width = 1,
Memory ports = 1,
Inorder = true,
Resources all types = 1.

g) Create aliases for sim-outorder and pipeview.pl by executing the following commands. Pipeview.pl is a perl script that helps in visualizing pipeline trace.

```
labs/ alias soo /home/simplescalar/simplesim-3.0/sim-outorder
```

```
labs/ alias pv /home/simplescalar/simplesim-3.0/pipeview.pl
```

h) Run sim-outorder with the config_a.cfg file by executing the following Command

```
labss/ soo –config config_a.cfg –ptrace config_a.trc 0:1024 –redir:sim sim_configa.out
./test-math
```

i) You should be able to see the output of the test-math file in the command window which means successful execution of the simulation.

j) View the simulation results by viewing the sim_configa.out file.

Now Answer the following Questions.

Q1: Describe the configuration.
Q2: What is the IPC (Instructions per cycle) of the test-math program using this configuration?
Q3: Is the forwarding implemented? To answer this question, consider the following two instructions

```
 ag:    sll     r2,r16,2
ah:    addu   r3,r3,r2
```

You can see RAW hazard between ag and ah instructions. This means that you can not execute ah until ag has written r2 into the register file. So without forwarding it is not possible to execute ah in the execution phase with ag being in the WB phase.

View the pipeline trace config_a.trc by executing the following command and see if you can find the two instructions such that ag is in the WB stage and ah is in the EX stage.

```
labss/ pv config_a.trc | less
```

2) In this part, we shall change the configuration. Every thing else remains the same as in part 1)

a) For this, make a copy of default.cfg, call it config_b.cfg. Make changes as in part 1) except that

Inorder = false

b) Run sim-outorder with the config_b.cfg file.

Now Answer the following questions

Q1: Describe the configuration.
Q2: What is the IPC (Instructions per cycle) of the test-math program using this configuration?
Q3: Compared to part 1) or configuration A, what can you say about this configuration?
Q4: The commit stage CT reorders the execution of instructions, explain why is it necessary?